Bachelor's degree in Mathematics

Bachelor's degree in Computer Engineering

Bachelor's degree thesis

High Frequency Trading via Deep Recurrent Neural Networks

Roger Romero Morral

May 2020

Supervised by Daniel P Palomar

Tutored by Luis Antonio Belanche

First of all, I would like to thank Prof. Daniel P Palomar for giving me the opportunity of doing by bachelor's thesis under his supervision and for all the help that he has given me during this months.

I want to thank Miguel for working with me in some of the topics of this thesis, and thanks to each one of the members of Prof. Daniel's research group for their interest on my work, coming to our meetings and sharing their advices and opinions, and for making me feel as part of the group since the first day.

I want also to thank Fundació Cellex and all the CFIS team members for this amazing opportunity of studying a double degree at UPC, finishing with this international research experience.

Finally, thanks to my family for all the support and help that they have given me during all this years.

Abstract

High frequency trading (HFT) is a type of financial trading that takes place on the order of minutes, as opposed to lower frequency trading like daily trading or monthly trading. Given the high frequency of decisions, it can only be executed in an automated way with algorithms, termed algorithmic trading. Decisions have to be made based on the past history, given in the form of snapshots of the Limit Order Book (LOB) which consists on several levels of the pending bid and ask orders. It is not clear what are the key features that should be used for decision making. Practitioners usually handcraft some features based on their experience and then use some simple linear regression for the forecast. The purpose of this project is to bypass the need for handcrafting features and fully use the potential of deep neural networks to take the raw data as input. Since the data naturally follows a time sequence, we will make use of the widely successful recursive neural networks (RNN) like the long short term memory (LSTM) architecture.

Keywords

Machine learning, Deep Learning, Recurrent Neural Networks, Long Short-Term Memory Networks, Quantitative Finance, High Frequency Trading

AMS Code

68T01

Contents

Chapter 1

Introduction

High Frequency Trading (HFT) is a type of trading in which the investment decisions have short investment horizons, typically minutes, seconds or even fractions of a second. Due to the high speed of decisions, it can only be executed in an automated way with algorithms.

It has become a really important type of trading; it is estimated that in 2016 HFT on average initiated between 10% and 40% of trading volume in equities, and between 10% and 15% of volume in foreign exchange and commodities.

HFT algorithms work by looking at previous data to make the investment decisions. Most practitioners use complex algorithms that take into account handcrafted features from the raw financial data. It is not clear which features are the key ones, and often they are chosen based on the trader's experience. A widely used type of such algorithm is using linear regression to forecast the future prices and then make the decisions.

However, financial data is highly nonlinear, so it is natural to think of using nonlinear machine learning algorithms (as opposed to linear regression) to produce a better forecast of future prices. Important examples of nonlinear algorithms are **gradient boosting trees** (which we will use as a benchmark) and **neural networks**.

In particular, a **recurrent neural network** (RNN) is a type of neural network that somewhat keeps track of the previous inputs and uses them along with the current one to make predictions. Therefore, it is mostly suitable when working with time series in which the current value depends partially on past values. This makes RNNs good

candidates as HFT algorithms, as financial data naturally follows a time series.

1.1 Research goals

The goal of this thesis is to investigate if recurrent neural networks are appropriate for working with financial high-frequency data and using them in HFT algorithms without the need of handcrafted features, letting the network learn directly from the raw data. In particular, we will use a Long Short Term Memory (LSTM) network, which is a type of RNN that has been successful in many time-series forecasting tasks.

To check the accomplishment of our goal, we will simulate trading with a really simple strategy based on the predictions of the network and, once the simulations are done, we will use some trading performance measures to quantify the results. Finally, we will compare the results of the LSTM network to the ones obtained with other machine learning algorithms to check it makes sense to use RNNs instead of simpler algorithms.

1.2 Overview of next sections

The next sections of this thesis will be structured as following:

- In Chapter 2 we will give an introduction to the High Frequency Trading concepts and terminology, and a description of the dataset that we will use later on.
- In Chapter 3 we will talk about related work that has been done in the field of machine learning applied to High Frequency Trading.
- In Chapter 4 we will describe some quantitative finance concepts that will be useful when assessing our models performance.
- In Chapter 5 we will introduce the machine learning algorithms that will be used as benchmarks.
- In Chapter 6 we will talk about recurrent neural networks and LSTM networks, and present the LSTM architecture that we will use to accomplish the goal of the thesis and its results.
- In Chapter 7 we will discuss the conclusions of the thesis.

Chapter 2

Limit Order Book

The Limit Order Book (LOB) is one of the most important concepts in the High Frequency Trading world; algorithms make the decisions mostly based on its current and past status. More than half of the markets in today's highly competitive and fast-paced financial world now use a limit order book (LOB) mechanism to facilitate trade [1].

In this chapter we will formally describe the LOB as well as some other concepts that will also be important. We will also look at the LOB data that we will use later on to train and test our models.

2.1 Orders and Limit Order Book

We will begin by defining the conforming elements of the Limit Order Book: orders.

Definition 2.1.1. An **order** is an ordered pair $x = (p_x, \omega_x, t_x)$. It represents a commitment to sell (if $\omega_x > 0$) or buy (if $\omega_x < 0$) up to $|\omega_x|$ units of the traded asset at a price no less than (if it is a sell order) or no greater than (otherwise) p_x, registered at time t_x.

Orders are sent by traders to the exchange market, which will then process the order looking at their Limit Order Book:

Definition 2.1.2. A **Limit Order Book** (LOB) $\mathcal{L}(t)$ is the set of all active orders in a market at time t.

The active orders in a LOB $\mathcal{L}(t)$ can be partitioned into the set of active buy orders

$\mathcal{B}(t)$, for which $\omega_x < 0$, and the set active sell orders $\mathcal{A}(t)$, for which $\omega_x > 0$.

To process the order (p_x, ω_x, t_x), the exchange market acts as following:

- If it is a **buy order**: if there are active sell orders in the LOB with price less than p_x it will execute the corresponding trade. It will continue to do so until it reaches an $|\omega_x|$ trade volume or until there are no more possible trades.
- If it is a **sell order**: if there are active buy orders in the LOB with price greater than p_x it will execute the corresponding trade. It will continue to do so until it reaches an $|\omega_x|$ trade volume or until there are no more possible trades.
- It will remove from the LOB the orders that have been fulfilled.
- If the process has finished because there are no more possible trades, it will store the order (with the remaining ω_x after the trades have been executed) in the LOB.

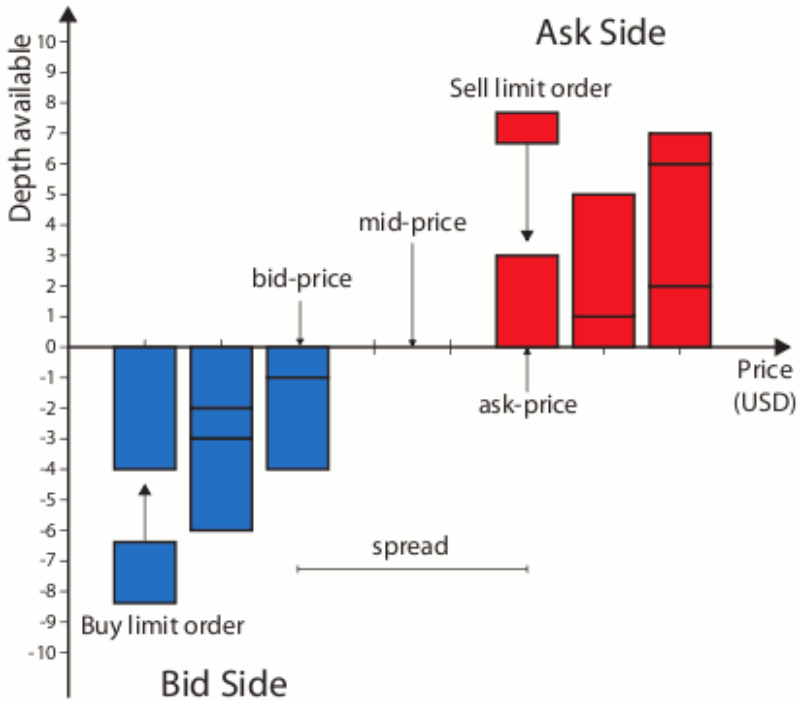

Figure 2.1: Graphical representation of a LOB

We will now define some concepts regarding the LOB that will be referred to frequently during the thesis.

Definition 2.1.3. The **tick size** π of a LOB is the smallest permissible price interval

between different orders within it. All orders must arrive with a price that is specified to the accuracy of π.

The tick size is important as it may have to be taking into account when designing a trading strategy.

Definition 2.1.4. The **bid price** at time t is

$$b_t := \max_{x \in \mathcal{B}(t)} p_x$$

Definition 2.1.5. The **ask price** at time t is

$$a_t := \min_{x \in \mathcal{A}(t)} p_x$$

From the definitions follows that if we have one unit of an asset and want to sell it instantly we will do it at the **bid price**, and if we want to buy one unit of the asset we will do it at the **ask price**.

In general, we will call $P_i^{bid}(t)$ and $P_i^{ask}(t)$ the bid and ask prices (respectively) at level i of the LOB; therefore, $b_t = P_1^{bid}(t)$ and $a_t = P_1^{ask}(t)$. We will also refer to the bid and ask volumes at level i as $V_i^{bid}(t)$ and $V_i^{ask}(t)$ respectively.

Definition 2.1.6. The **mid price** at time t is

$$m_t := \frac{a_t + b_t}{2}$$

The mid price is a really important concept in finance. Usually, when we do not need details about the bid and the ask price of an asset (for example, in long-term investing), we use the mid price as the asset's price.

2.2 The dataset

Now that we have defined a LOB and given some important definitions we will take a look at the data that we will use to train and test our models. First we will introduce the concept of a *futures contract*:

Definition 2.2.1. A **futures contract** is a legal agreement to buy or sell a particular commodity asset, or security at a predetermined price at a specified time in the future. Futures contracts are usually traded on a futures exchange.

The **CSI 300 Index Futures** are futures contracts that have as underlying asset the **CSI 300** index, a capitalization-weighted stock market index designed to replicate the performance of top 300 stocks traded in the Shanghai and Shenzhen stock exchanges. They are traded in the **China Financial Futures Exchange**, and its trading hours are from 9:30 to 11:30 and from 13:00 to 15:00.

Our dataset consists of 140 days of partial snapshots of the Limit Order Book of the **CSI 300 Index Futures** every 0.5 seconds. It contains the prices and volumes of the 5 first bid and ask levels of the LOB. The tick size of the LOB is 0.2.

We will use the first 100 days of data to train our models, the next 20 days to validate them and the last 20 days to simulate the trading and get conclusions.

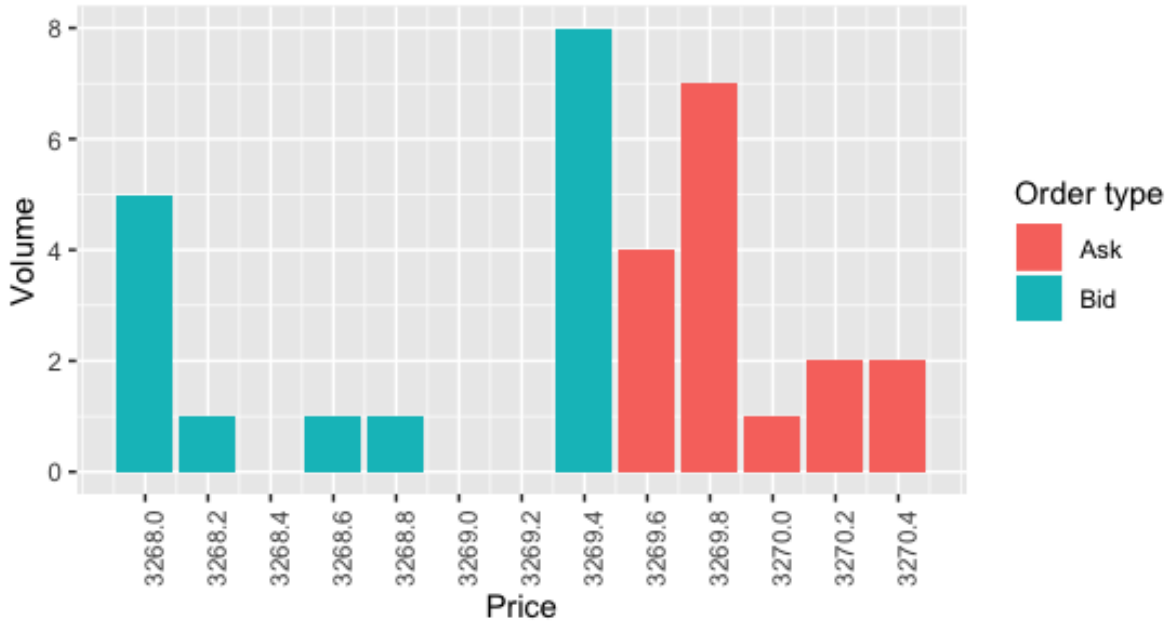

Figure 2.2: Snapshot of the LOB

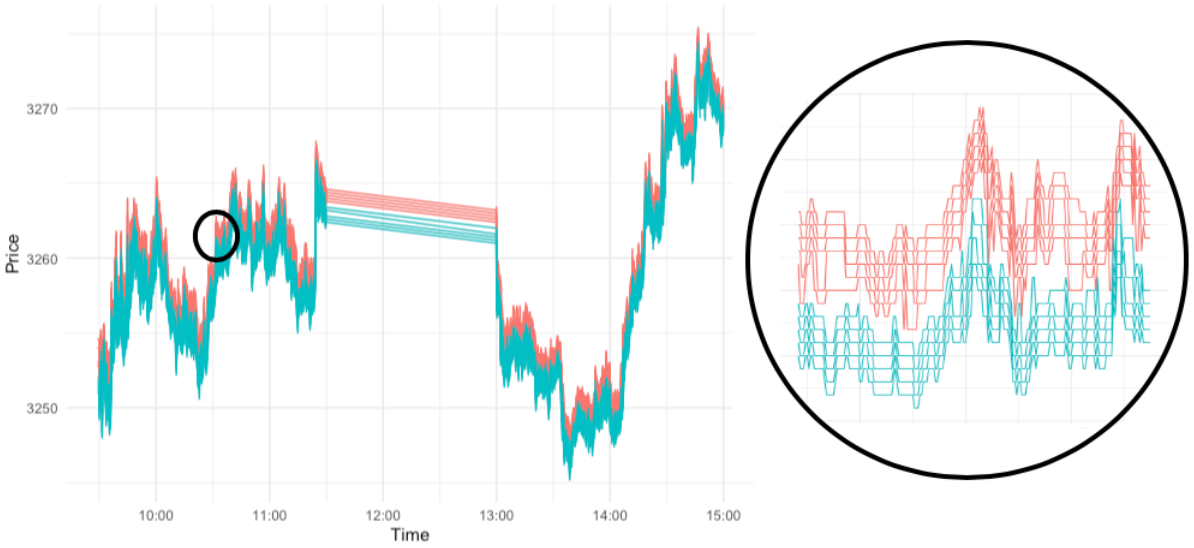

Figure 2.3: Graph of the 5 best ask and bid prices for a whole trading day

Chapter 3

Related work

In this chapter we will give an overview of previous research on machine learning methods applied to Limit Order Book data. This will give us an idea of how the problem is approached in the academic world before starting our own work. To be able to fully understand the related work on the topic we have to start by giving a simple introduction to machine learning.

3.1 Machine learning overview

Machine learning is the study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead.

It's based on building a mathematical model based on sample data, known as *training data*, in order to make predictions or decisions without being explicitly programmed to perform the task.

Machine learning has been developing for many years (the term **machine learning** was first used in 1959), and nowadays there are different types of learning (*supervised learning*, *unsupervised learning*, *reinforcement learning*...). Here, we will focus on **supervised learning**, which is the task of learning a function that maps an input to an output based on example input-output pairs.

In supervised learning, we assume that there exists a function $f : \mathbb{R}^n \to \mathcal{A}$ that truly maps any given input to the corresponding output correctly. However, this function

may be extremely difficult or impossible to find because of two reasons:

- We may not have information about some of the factors that affect the output.
- We have a finite set of input-output pairs so it could be impossible to know exactly the value of f for any input.

The goal of supervised learning is to build a *function approximator* $\hat{f} : \mathbb{R}^m \to \mathcal{A}$ (with $m \leq n$) using a set of input-output pairs $\{(\mathbf{x}_i, y_i)\}$ that we know to be correct.

There are many different ways to build $\hat{f}$, and each of them has its advantages and disadvantages. Some of the most popular supervised learning algorithms are Linear Regression, Support Vector Machines, Decision Trees and Neural Networks. In fact, these are classes of algorithms and each of them has different subtypes.

3.1.1 Regression and classification

Supervised learning problems are divided into two classes: regression and classification.

- **Regression** problems are the ones in which $\mathcal{A}$ (the set of possible outputs) is a continuous subset of $\mathbb{R}$. For example, predicting the price of a house based on its area and number of rooms is a regression task.
- **Classification** problems are the ones in which $\mathcal{A}$ is discrete. In classification tasks we try to map the input to the class to which it belongs. An example of a classification problem is predicting the gender of a person based on their voice pitch.

3.1.2 Supervised learning applied to LOB data

In our case, it seems natural to use the current and past status of the LOB as input to the algorithms; we will see different ways to do so.

However, it's more difficult to choose what to use as output: we can approach the predictions from a **regression** perspective and try to forecast the future ask, bid or mid prices; we can also take a **classification** approach and come up with a way to classify the future movements of the prices into categories.

In this thesis we will focus on the second type of approach as we will exploit the advantages of using classification in our trading strategy. However, the regression perspective is also very interesting and has been object of research in the past years. In

the book Econometrics of Financial High-Frequency Data [2], Hautsch describes the use of different regression models in LOB forecasting.

The idea of using **classification** with LOB data is to label each time step, so when we run our model in a trading environment we will produce a prediction each time we get new data (in our case, every 0.5 seconds). Then, a trading algorithm will make a decision based on this prediction.

3.2 Related work

There is also a wide variety of research on the classification scenario. As explained, when training a classification machine learning algorithm one has to define two sets: the set of features that will be used as input to the algorithm and the set of possible outputs.

3.2.1 Feature set

In 2015, Kercheval and Zhang released a paper in which they used a handcrafted set of features from the LOB as input to a Support Vector Machine [3]. This paper has been of key importance to the topic as most of the posterior researchers have also used the same features as input to their algorithms.

It is important to note that they used a different type of dataset than ours: while we have a row of data every 0.5 seconds, their dataset consisted of a row of data each time an order arrived to the market. This makes some of the features that they used impossible to obtain from our raw data.

The set consisted of three different subsets:

- A **basic set**, containing the ask and bid prices and volumes.
- A **time-insensitive set**, containing some features that only take into account the current status of the LOB.
- A **time-sensitive set**, containing some features that take into account both the current and past status of the LOB.

The full set of features is described in the following table:

Basic Set	Description(i = *level index*, $n = 10$)
$v_1 = \{P_i^{ask},\ V_i^{ask},\ P_i^{bid},\ V_i^{bid}\}_{i=1}^{n}$,	*price and volume (n levels)*

Time-insensitive Set	Description(i = *level index*)
$v_2 = \{(P_i^{ask} - P_i^{bid}), (P_i^{ask} + P_i^{bid})/2\}_{i=1}^{n}$,	*bid-ask spreads and mid-prices*
$v_3 = \{P_n^{ask} - P_1^{ask}, P_1^{bid} - P_n^{bid}, \lvert P_{i+1}^{ask} - P_i^{ask}\rvert, \lvert P_{i+1}^{bid} - P_i^{bid}\rvert\}_{i=1}^{n}$,	*price differences*
$v_4 = \{\frac{1}{n}\sum_{i=1}^{n} P_i^{ask},\ \frac{1}{n}\sum_{i=1}^{n} P_i^{bid},\ \frac{1}{n}\sum_{i=1}^{n} V_i^{ask},\ \frac{1}{n}\sum_{i=1}^{n} V_i^{bid}\}$,	*mean prices and volumes*
$v_5 = \{\sum_{i=1}^{n}(P_i^{ask} - P_i^{bid}),\ \sum_{i=1}^{n}(V_i^{ask} - V_i^{bid})\}$,	*accumulated differences*

Time-sensitive Set	Description(i = *level index*)
$v_6 = \{dP_i^{ask}/dt,\ dP_i^{bid}/dt,\ dV_i^{ask}/dt,\ dV_i^{bid}/dt\}_{i=1}^{n}$,	*price and volume derivatives*
$v_7 = \{\lambda_{\Delta t}^{la},\ \lambda_{\Delta t}^{lb},\ \lambda_{\Delta t}^{ma},\ \lambda_{\Delta t}^{mb},\ \lambda_{\Delta t}^{ca},\ \lambda_{\Delta t}^{cb}\ \}$	*average intensity of each type*
$v_8 = \{\mathbf{1}_{\{\lambda_{\Delta t}^{la} > \lambda_{\Delta T}^{la}\}},\ \mathbf{1}_{\{\lambda_{\Delta t}^{lb} > \lambda_{\Delta T}^{lb}\}},\ \mathbf{1}_{\{\lambda_{\Delta t}^{ma} > \lambda_{\Delta T}^{ma}\}},\ \mathbf{1}_{\{\lambda_{\Delta t}^{mb} > \lambda_{\Delta T}^{mb}\}}\}$,	*relative intensity indicators*
$v_9 = \{d\lambda^{ma}/dt,\ d\lambda^{lb}/dt,\ d\lambda^{mb}/dt,\ d\lambda^{la}/dt\}$,	*accelerations(market/limit)*

Figure 3.1: Table of features introduced by Kercheval and Zhang

In our case, of the time-sensitive features, we can only obtain the ones in v_6, where the average time derivatives are computed over the most recent 1 second. Obviously, we can obtain all the features in the basic and time-insensitive set.

More recently, in 2019, Ntakaris et. al. released a paper [4] in which they compared different sets of features to predict mid-price movement:

- A set containing only **econometric** features.
- A set containing only **technical and quantitative** indicators (Tech&Quant).
- The set described in Figure 3.1.
- A set of features extracted by an LSTM autoencoder.

In this paper, the Tech&Quant indicators outperform the rest of them, followed closely by the ones in Figure 3.1. However, most of the Tech&Quant indicators are either impossible or really difficult to implement with our dataset (as most of them have parameters that require heavy tuning). For this reason, we will use the set of features in Figure 3.1.

3.2.2 Output set

In [3], Kercheval and Zhang describe two ways to classify a LOB entry at time step t.

- **Mid price movement**
  - If $m_t > m_{t-1}$: we label time step t as an **upwards mid price movement**.
  - If $m_t < m_{t-1}$: we label time step t as a **downwards mid price movement**.
  - Otherwise: we label time step t as **no movement**.
- **Spread crossing**
  - If $b(t+t') \geq a(t)$ for some $t' \leq h$ (h is a horizon constant that we have to choose): we label time step t as an **upwards spread crossing**.
  - If $a(t+t') \leq b(t)$ for some $t' \leq h$: we label time step t as a **downwards spread crossing**.
  - Otherwise: we label time step t as **no spread crossing**.

In the past years, most researchers have been focused in predicting **mid price movement** or similar labels such as smoothed mid price movement, as in [5], [6] or [7], and measuring the models performance using machine learning measures such as accuracy, precision or F1-score.

However, it is important to note that mid price movement is a statistical indicator (though not a guarantee) of potential trading profits. In contrast, bid-ask spread crossing is a less-frequent occurrence that however does assure a profit if correctly identified in advance. In this thesis, we will compare both labels and choose the ones that best fit our interests.

Also, we will not use machine learning measures to assess our models performance: instead, we will use quantitative finance measures, as we think that the ultimate goal for these models is not achieving good accuracy or precision (as in other machine learning fields), but earning money by using the predictions in a trading strategy.

In January 2020 the paper [7] was published. In this paper, the authors present a complex deep neural network architecture combining convolutional and LSTM layers to predict mid price movement. Although in this thesis we will not use such architecture as it is beyond the goal of the thesis, it could be really interesting to try to apply it to the spread crossing problem with our dataset.

Finally, we have to take into account that we should not use the results achieved by other researchers as baselines to our model, because they use not only a different dataset

but also a different type of LOB data. Instead, we have to adapt their work to our dataset and build our own benchmark models.

Chapter 4

Financial theory and performance measures

In this chapter we will introduce some quantitative finance concepts that will be important to measure the performance of our models. Then, we will describe a really basic trading strategy that we will use to simulate the trading using our models predictions.

We will base the definitions of this chapter in the ones given in the course **Portfolio Optimization with R** given by Prof. Daniel P. Palomar in the MSc in Financial Mathematics, HKUST [8].

4.1 Quantitative finance basics

Let p_t be the price of an asset at (discrete) time index t. We can choose the difference between consecutive time indexes depending on our interests. For instance, in portfolio optimization it is usual to use daily data, whereas in High Frequency Trading the differences between time indexes can be of minutes, seconds or even less (in our case, 0.5 seconds).

Definition 4.1.1. The **Simple return** (a.k.a. linear or net return) is at time t is

$$R_t = \frac{p_t - p_{t-1}}{p_{t-1}} = \frac{p_t}{p_{t-1}} - 1$$

.

Returns (and log-returns, which are defined as $r_t = \log(1 + R_t)$) play a key role in portfolio optimization; investors use optimization methods to choose the best allocation of their budget to the assets by minimizing an objective function that usually takes the returns of different assets as input (among other things).

For example, if we take the mid-price as the price of our data (which is not accurate but is enough when we are taking daily measurements), these are the daily returns for the first 100 days:

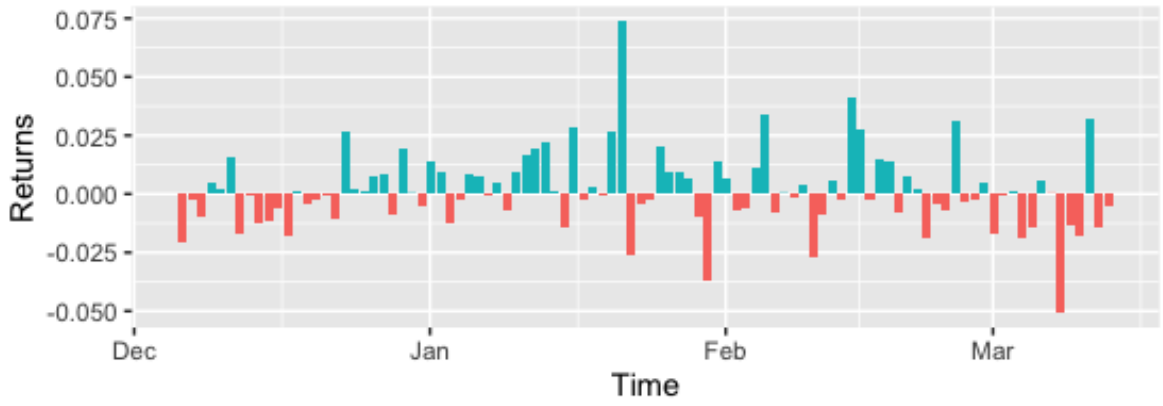

Figure 4.1: Daily returns of the our data for the first 100 days

However, in our context (HFT) it does not make sense to use returns in the same way. What makes the use of returns interesting for us is that, changing the definition of p_t to be the **total money that we have at time step** t, we can apply all the existing theory on quantitative metrics to assess our model performance.

The most basic performance measure of an investment is the **average return**: the mean of the returns of which we have a record. We will refer to the average return as μ.

We can compare different trading strategies by simulating them on the same environment and, obviously, the one with higher average return will be the one that more money makes (*on the data that we have, we can't know if it will continue to be like this because we don't see the future!*).

However, we can't rely only on average returns to compare the performance of different strategies; we have to also take into account the risks that we are taking with each

strategy. For example, imagine the following scenario:

- **Strategy 1** yields an average 10% annual return.
- **Strategy 2** yields an average 20% annual return, but its probability of going bankrupt is twice as much as the one in **Strategy 1**.

Obviously, we can't directly say that, as it has higher returns, **Strategy 2** is the best one; it is also more risky.

An investor deciding between these two strategies will have to choose the one that fits best their interests. Some investors may be more risk-averse and then they would choose **Strategy 1**, whereas other investors would be willing to take more risks to achieve higher returns and thus choose **Strategy 2**.

At this point, it's clear that we need a way to measure the risk of an investment strategy. Now we will describe different ways to do this from a quantitative point of view.

4.1.1 Risk control

The most basic measure of risk is given by the **variance of the returns** (Markowitz 1952 [9]): a higher variance means that there are large peaks in the distribution which may cause a big loss. We will refer to the standard deviation of the returns (the square root of the variance) as σ.

To illustrate this, we will use artificial data: consider the following two strategies:

- **Strategy 1**, with $\mu_1 = 2$ and $\sigma_1 = 1$.
- **Strategy 2**, with $\mu_2 = 4$ and $\sigma_2 = 3$.

For the sake of simplicity, we will assume that returns follow a Gaussian distribution, although in real life data this is usually false. The following graph shows the histogram of returns of the two strategies using 10000 samples of randomly generated data.

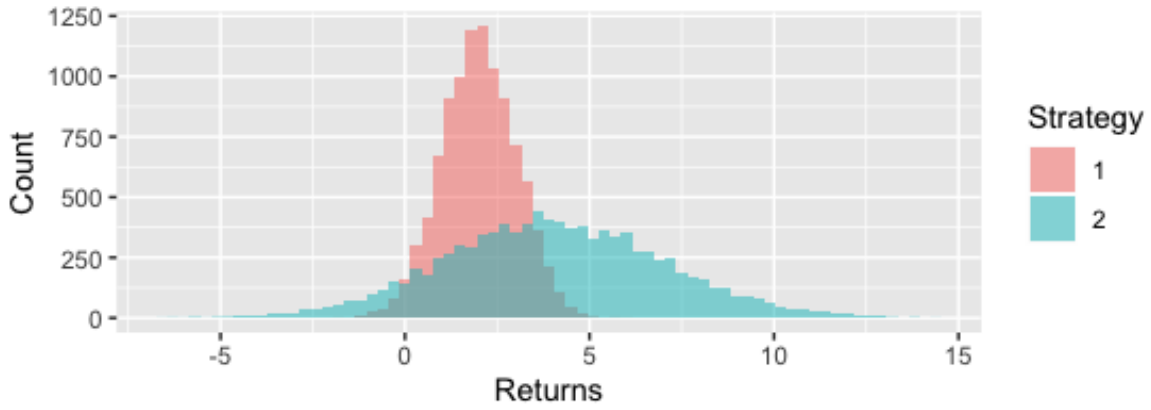

Figure 4.2: Histogram of returns of the two strategies

As we can see, if we focus only in the losses, **Strategy 2**, which has higher σ, has a bigger number of losses.

This example shows the fact that the strategy with higher σ is usually the most risky, although it can also give greater benefits.

However, **variance** is not a good measure of risk in practice: its main disadvantage is that it penalizes both the unwanted high losses and the desired high earnings, and it also has other issues when used in portfolio optimization.

Next, we will describe more meaningful measures for risk than the variance.

4.1.1.1 Downside risk

The idea of **downside risk** is that the left hand side of the return distribution involves risk while the right hand side contains the better investment opportunities.

An important type of **downside risk** measure is the **downside deviation**, which is a special case of the more general **lower partial moments (LPM)**.

Definition 4.1.2. Let R be the random variable representing the returns of our strategy. The **lower partial moments (LPM)** are defined as

$$\text{LPM} = \mathbb{E}[((\tau - R)^+)^\alpha]$$

where $(\cdot)^+ = \max(0, \cdot)$.

The parameter τ is termed the **disaster level**, and the parameter α reflects the investor's feeling about the relative consequences of falling short of τ by various amounts (the grater α is, the more risk-averse is the investor).

In particular, setting $\tau = 0$ and $\alpha = 2$ and taking the square root, we get the **downside deviation**

$$\sigma_d = \sqrt{\mathbb{E}[((-R)^+)^2]}$$

Obviously, the greater the downside deviation, the more risky our investment strategy is.

4.1.1.2 Drawdown

The **drawdown (DD)** at time t is the decline from a historical peak of the cumulative profit.

Definition 4.1.3. Let $X(t)$ be the cumulative profit at time t. Then,

$$D(t) = \frac{\mathrm{HWM}(t) - X(t)}{\mathrm{HWM}(t)}$$

where

$$\mathrm{HWM}(t) = \max_{1 \leq \tau \leq t} X(t)$$

We can use the drawdown plot to compare the risk of different strategies in a visual way.

4.1.2 Sharpe Ratio

Now that we know how to quantify the profit of a strategy (with the average returns) and we know some ways to measure its risk, it seems natural to use a formula that measures the *general wellness* of a strategy by combining in an appropriate way profits and risk.

Then, we can compare between different strategies simply by inputting their returns on this formula.

Sharpe Ratio (SR) [10] is one of such formulas, and it has a lot of popularity among

researchers and also in the industry. It is defined as

$$SR = \frac{\mu - R_f}{\sigma}$$

where R_f is the risk-free rate, the return that we can get without taking any risk (for example, buying US treasury bonds). In High Frequency Trading, $R_f = 0$ as there doesn't exist a risk-free investment with short time horizon.

Informally, **the Sharpe Ratio measures how much risk we are taking for unit of benefit**.

Recall our example, in which we had two strategies:

- **Strategy 1**, with $\mu_1 = 2$ and $\sigma_1 = 1$.
- **Strategy 2**, with $\mu_2 = 4$ and $\sigma_2 = 3$.

In this example, **Strategy 1** has a SR of 2 and **Strategy 2** has a SR of $\frac{4}{3}$. So, in terms of Sharpe Ratio, **Strategy 1** is better than **Strategy 2**.

We may be tempted to say that **Strategy 1** is always better than **Strategy 2**; this is false, as an investor who is not happy with a return of 2 would prefer **Strategy 2** although it doesn't optimize the return per risk.

Sharpe Ratio depends on the time scale that we are using: if we use daily returns we will get a different SR than if we use annual returns for the same trading strategy. For this reason, we define **annualized Sharpe Ratio** as

$$ASR = \sqrt{N}\frac{\mu - R_f}{\sigma}$$

where N is the number of trading periods in a year. For example, if we are using daily returns, $N = 252$, the number of trading days in a year.

In our case, we will use 20 days of data to test the models, and we have data every 0.5 seconds. As the trading hours for the **China Financial Futures Exchange** are from 9:30 to 11:30 and from 13:00 to 15:00, then $N = \frac{252}{20} \cdot \frac{4 \cdot 60 \cdot 60}{0.5} = \frac{252}{20} \cdot 28800$.

In traditional investing, an annualized Sharpe Ratio greater than 1 is considered acceptable to good, a ratio higher than 2 is rated as very good, and a ratio of 3.0 or higher is considered excellent.

However, in HFT Sharpe Ratio makes little sense as we can achieve a really high Sharpe Ratio while keeping our earnings really low. To illustrate this, consider the following example:

- We start with 10000 RMB.
- During a whole trading day, we perform only one operation, buying at 3260 and selling at 3260.2 in the next 0.5 seconds.
- At the end of the day, we will have 10000.2 RMB.

Our returns vector will consist of 28799 0's and an occurrence of $\frac{0.2}{10000}$. If we calculate its annualized Sharpe Ratio we will get 15.87, which is really high in terms of traditional investing. However, we have only earned 0.2 RMB in a whole day!

This happens because Sharpe Ratio only takes into account the mean of the returns, not the total money earned. In HFT, returns are really low as prices don't change much in a time of 0.5 seconds, but this means that we are also taking less risk, so we can get a high Sharpe Ratio while earning a tiny amount of money.

Also, Sharpe Ratio uses variance as a risk measure, and we know that this is not accurate as we are penalizing big gains. This problem was addressed by Dr. Frank Sortino in 1994 [11] when he introduced the **Sortino Ratio**, which is defined as

$$SR = \frac{\mu - R_f}{\sigma_d}$$

where σ_d is the **downside deviation** of the returns. As said before, in High Frequency Trading we have $R_f = 0$.

To overcome these two issues we will use a modified version of the Sortino Ratio in which, instead of the mean of the returns, we use the sum of the returns taken only when we are *in* the market:/newline
Definition 4.1.4. Let R'_t be the returns of our strategy taken only when we are holding a position (he have at least one asset in possession) at time t. Then, we define the **Modified Sortino Ratio (MSR)** as

$$\text{MSR} = \sqrt{N}\frac{\sum R'_t}{\sigma_d}$$

Hence, we are taking into account the total money that we earn, not the mean of the

returns, while also taking into account the risk of our investments in an appropriate way.

4.2 Trading Strategy

Before starting with the machine learning part of the thesis, in which we will give some benchmark results that we will then compare to the LSTM results, we have to define the trading strategy that we will use once we get a prediction.

At each time step, our trading strategy will have as input a signal that can have 2 possible values: *buy the asset* or *do nothing*.

The trading strategy is described in the following pseudocode.

for timestep t **do**
  Signal $\leftarrow GetSignal()$
  if Signal = Buy **then**
    Buy one unit of the asset at price $a(t)$
    for i in 1 to h **do**
      if $b(t+i) > a(t)$ and not already sold **then**
        Sell one unit of the asset at price $b(i)$
      end if
    end for
    if not already sold **then**
      Sell one unit of the asset at price $b(t+h)$
    end if
  end if
end for

Where h is the horizon that we choose. Informally, each time that we buy one unit of the asset we sell it when:

- We can sell the asset for more than what we bought
- The horizon has passed

This strategy is really simple and simulating it may be unrealistic as we aren't taking into account *transaction costs* and *market impact*. However, it is good enough for the goal of this thesis, as we want to check if an LSTM network can be useful in a High

Frequency Trading environment; if we can make money using an LSTM network and this really simple strategy, then it can also be useful with a more sophisticated and realistic trading strategy that also takes into account other information.

As stated in Chapter 3, we will use both **mid-price movement** and **spread crossing** as signals and compare them. In particular, we will use **smoothed mid-price movement**, that we will describe in the next chapter.

We will simplify the labels described in [3] by merging the **downwards movement** class into the **no movement** one; we will do the same for the *spread crossing* labels.

Our signals will be:

- **Mid price movement**
  - **Buy** if we predict **upwards mid-price movement**.
  - **Do nothing** if we predict **no movement**.
- **Spread crossing**
  - **Buy** if we predict **upwards spread cross**.
  - **Do nothing** if we predict **no spread cross**.

It is clear that, with our strategy, predicting right a **spread cross** we will always make money. Although this is not the case with **mid-price movement**, it is still interesting to try to use it as labels because it may be easier to predict and lead to better performance.

Chapter 5

Machine learning benchmarks

In this chapter, we will use some machine learning algorithms to predict both *mid price movement* and *spread crossing*. We will use these predictions with our trading strategy to choose the set of labels that fits best our goal, and also as benchmarks for our LSTM predictions.

5.1 Overview of the algorithms

First, we will briefly describe the machine learning algorithms that we will use. We will not dive into much detail as they intend to serve just as baselines.

5.1.1 Logistic regression

Logistic regression is a generalization of **linear regression** in which instead of predicting a continuous value we predict the probability of the sample belonging to a given class. In our case, we want to predict the probability of a sample belonging to the class **upwards** (either *mid price movement* or *spread crossing*).

Let p be the probability of a given sample S, with feature vector x, belonging to the target class. When using logistic regression, we assume a linear relationship between x and the **log-odds** of the event S belongs to the target class. The **log-odds** are defined as $l = \frac{p}{1-p}$. Therefore, we assume the following relationship:

$$l = \alpha + \beta^T x$$

where α and β are inferred from the data.

Once we know α and β, we can get the probability of a new sample (with feature vector x') belonging to the target class:

$$p' = \frac{e^{\alpha+\beta^T x'}}{1 + e^{\alpha+\beta^T x'}}$$

We will use LASSO logistic regression, which adds a penalty to the L^1-norm of β in order to produce a more robust and explainable model.

5.1.2 Random Forest

Random Forests are ensemble methods that use multiple weak learners (decision trees) to produce a stronger output.

Informally, a **decision tree** is a tree-like graph with nodes representing the place where we pick a feature and ask a question; edges represent the answers the to the question; and the leaves represent the actual output (the probability of belonging to the target class).

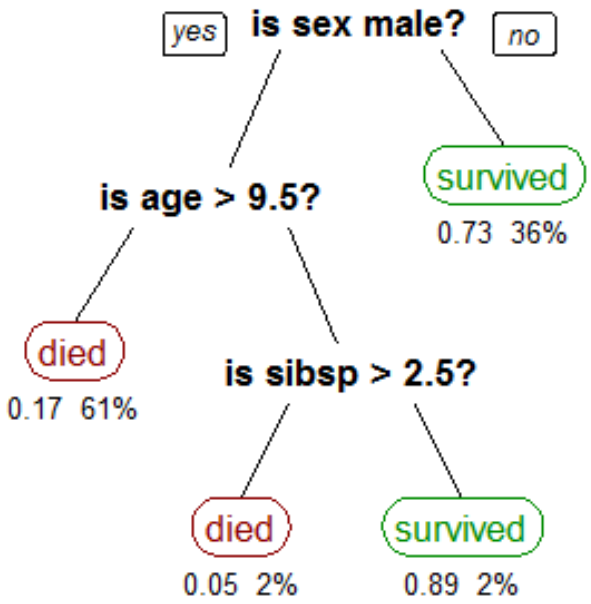

Figure 5.1: Decision tree that classifies survival on the Titanic. The figures under the leaves show the probability of survival and the percentage of observations in the leaf.

Decision trees have some problems, such as being **non-robust** (a small change in the training data can result in a large change in the tree) and **prone to overfitting** to the training data. Random Forests help overcome these limitations by applying the **bagging** technique to decision tree learners.

Bagging consists in training B different trees, each one trained on a different subset of the dataset (the sampling is done with replacement, so a sample can be in more than one subset).

To predict the probability of a new sample S belonging to the target class, we average the outputs produced by each of the B trees on the input S.

5.1.3 Gradient boosting trees

Like random Forests, **gradient boosting trees** are ensemble algorithms that use decision trees as the weak learner. However, they do not use a set of decision trees; instead, when using a gradient boosting tree, we start with a single decision tree and

iteratively improve it using gradient descent over a certain loss function, getting a stronger decision tree at each iteration, and stopping when convergence is achieved (or after a certain number of steps).

Gradient boosting trees have become really popular in the past years because of their great performance. In fact, in most machine learning competitions nowadays gradient boosting trees are the only models that can give results similar to deep learning algorithms.

One of the most famous **gradient boosting trees** implementations is called **XGBoost**, which is the one that we will use in this thesis.

5.1.4 Multilayer perceptron

The **multilayer perceptron (MLP)** is one of the most simple types of **feed-forward artificial neural network**. As LSTM networks are also neural networks, we are going to dive further into detail into the explanation of MLPs because they share some characteristics with LSTMs.

Definition 5.1.1. A **neuron** is a computing unit that consists of:

- An input $x(t)$
- An internal state $s(t)$ (the *memory*)
- A parameter set w
- An output $f(t)$

With t being the discrete time index.

The general shape of a neural network is a labeled directed graph. If the network is recurrent, the graph contains cycles. When it is not recurrent, it is called **feed-forward**.

Feed-forward networks consist of neurons that have:

- An input x
- A parameter set w
- An output f

With $f(x) = g(\phi(x, w))$. We call ϕ the **aggregation function**, and g the **activation function**.

If we set

- $\phi : \mathbb{R}^d \times \mathbb{R}^{d+1} \to \mathbb{R}$, $\phi(x, (w, w_0)) = w^T x + w_o$
- g a non-decreasing monotonic function

the resulting model is called a **perceptron**.

Important examples of activation functions are the **sigmoid** function $g(z) = \frac{1}{1+e^{-z}}$ (used in two-class classification), the *tanh* function $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ or the **ReLu** function $g(z) = \max(0, z)$.

We can generalize the perceptron to have multiple outputs ($f : \mathbb{R}^d \to \mathbb{R}^m$) by setting $f_k(x) = g(w_k^T x + w_{k0})_{k=1,\dots,m}$, where f_k is the k-th coordinate of the output.

The parameters w of a perceptron are trained by minimizing a loss function over the training dataset $T = \{(x_1, y_1), \dots, (x_n, y_n)\}$. The choice of the loss function varies depending on the goal. In our case, for two-class classification the output will be the probability of a sample belonging to the target class, and the loss function will be the **discrete cross entropy function**:

$$L(w) = -\sum_{i=1}^{n} \left(y_i \log(f(x_i)) + (1 - y_i) \log(1 - f(x_i))\right)$$

With $y_i = 1$ if the sample i belongs to the target class and $y_i = 0$ otherwise. This optimization is done via **gradient descent** starting with a random parameter set; obtaining the gradient is straight-forward from the loss function definition.

Definition 5.1.2. A **multilayer perceptron** is the model that results from **successively composing perceptrons**.

If we subtract one from the number of perceptrons used we get the **number of hidden layers**. For example, if we have three perceptrons $f_1 : \mathbb{R}^d \to \mathbb{R}^{h_1}$, $f_2 : \mathbb{R}^{h_1} \to \mathbb{R}^{h_2}$ and $f_3 : \mathbb{R}^{h_2} \to \mathbb{R}$, then $f_3 \circ f_2 \circ f_1 : \mathbb{R}^d \to \mathbb{R}$ is an MLP with two hidden layers.

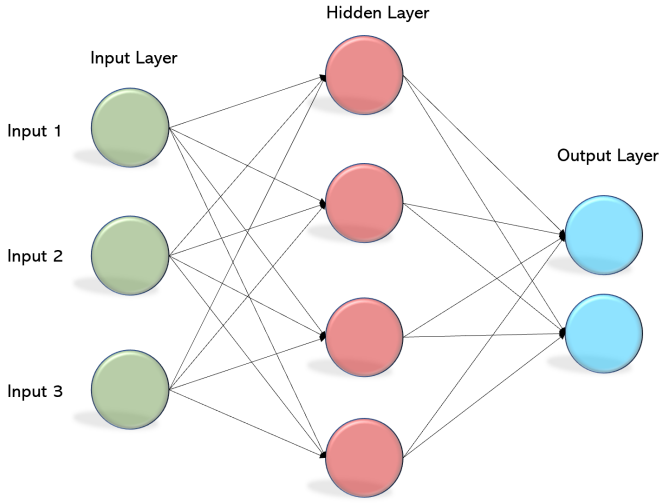

Figure 5.2: Representation of an MLP with one hidden layer

Multilayer perceptrons are also trained using **gradient descent**. However, obtaining the gradient is not as easy as with the perceptron, and it is done via the **backpropagation algorithm**, which works by computing the gradient of the loss function with respect to each weight using the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule.

In our case, we will use as benchmark an MLP with one hidden layer consisting of 16 units.

5.2 Training framework

Each time we train and test a model we are going to follow the next steps:

1. Train the model using the first 100 days of data.
2. Use the next 20 days of data to choose the best probability threshold t. Given a new sample S, we are going to predict that it belongs to the target class ($y_S = 1$) if the outputted probability is greater than t. Note that t may vary from model to model.
3. Simulate trading using our models predictions as the trading strategy signals (with the chosen t as the threshold) in the last 20 days of data.

We are going to choose the t that maximizes the Modified Sortino Ratio (MSR) over the validation period.

5.3 Mid-Price movement and Spread Crossing

In this section we are going to compare between the two types of labels that are most used by researchers on this topic.

Let (x_t, y_t) be the sample at time step t (recall that x_t is the set of features introduced in [3], described in Chapter 3).

Once we choose a horizon h, we will define the labels as following:

Mid price movement

$$y_t = \begin{cases} 1 & \left(\frac{1}{h}\sum_{i=1}^{h} m(t+i)\right) - m(t) > \tau \\ 0 & \text{otherwise} \end{cases}$$

τ is a parameter that we can choose. In our case, we are going to use a different τ for each day. For a given day d we will do the following:

- Let m_d be the mean of $m(t)$ over days $d-1, \ldots, d-5$.
- τ for day d will be $\frac{0.4}{m_d}$

In this way, we are setting $y_i = 1$ only when there is a significant upwards movement in mid prices (as the tick size is 0.2), and therefore trading with these signals will have more probabilities of giving benefits.

Spread crossing

$$y_t = \begin{cases} 1 & b(t+t') \geq a(t) \text{ for some } t' \leq h \\ 0 & \text{otherwise} \end{cases}$$

Note that the horizon h is a parameter that we also have to optimize. It makes sense then to train the models using different h and see the results with each one.

To compare the two type of labels, we will follow our *training framework* with each of the models described above for $h = 10, 20, 30, 40$. There are models that are so bad that

they always lose money, so the optimization algorithm outputs $t = 0.99$ to avoid doing any trade and keep the losses at 0. This would not let us compare well between these models and the other ones.

Also, as we are optimizing the Modified Sortino Ratio, if the model makes no losses at all then the *downside risk* is 0 resulting in an "infinite" MSR. However, we don't want this because making no losses means predicting 1 a tiny amount of time, so the returns would be negligible.

To avoid these two issues we are going to optimize only over the t's that make the model predict 1 at least 1000 times over the validation data.

These are the results for each value of h (in *red* the *Mid Price movement* results, and in *blue* the *Spread Crossing* results):

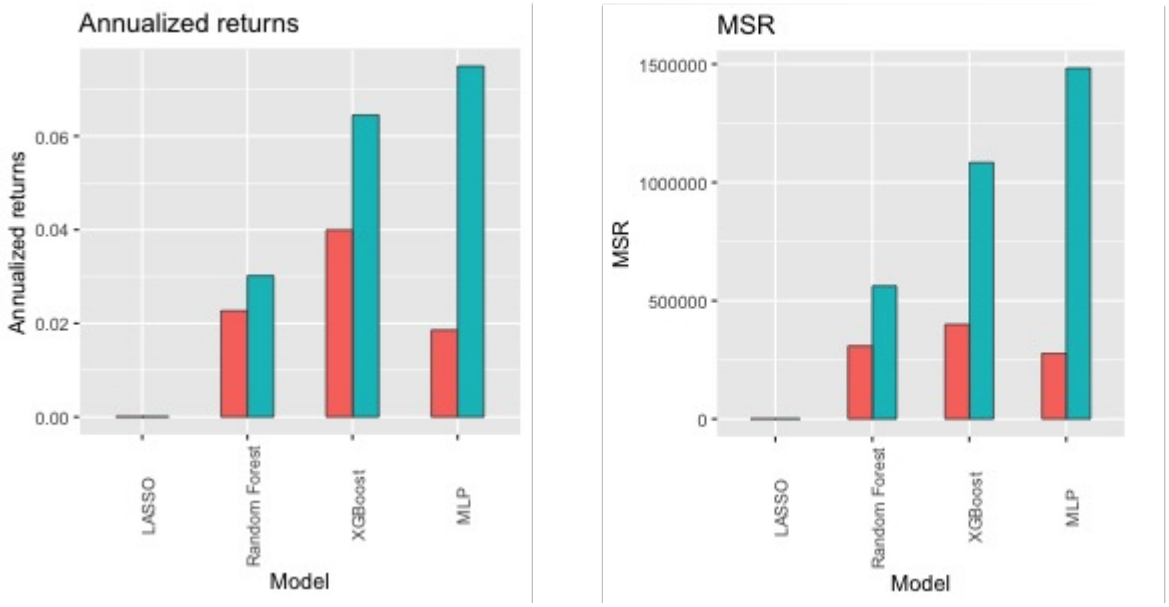

Figure 5.3: Results for $h = 10$

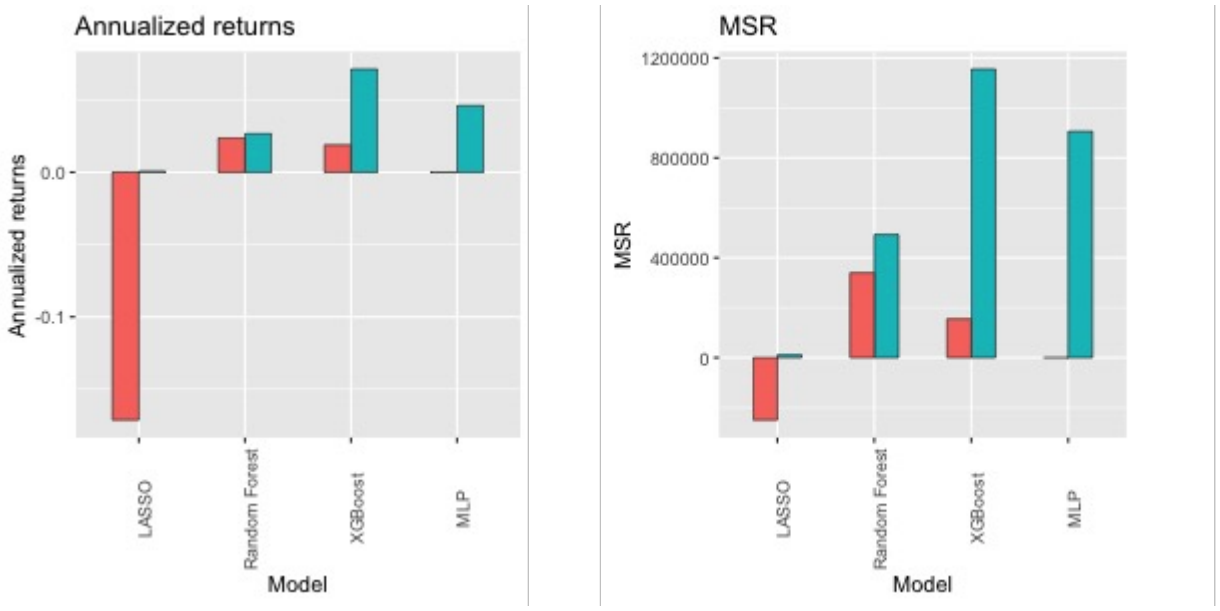

Figure 5.4: Results for $h = 20$

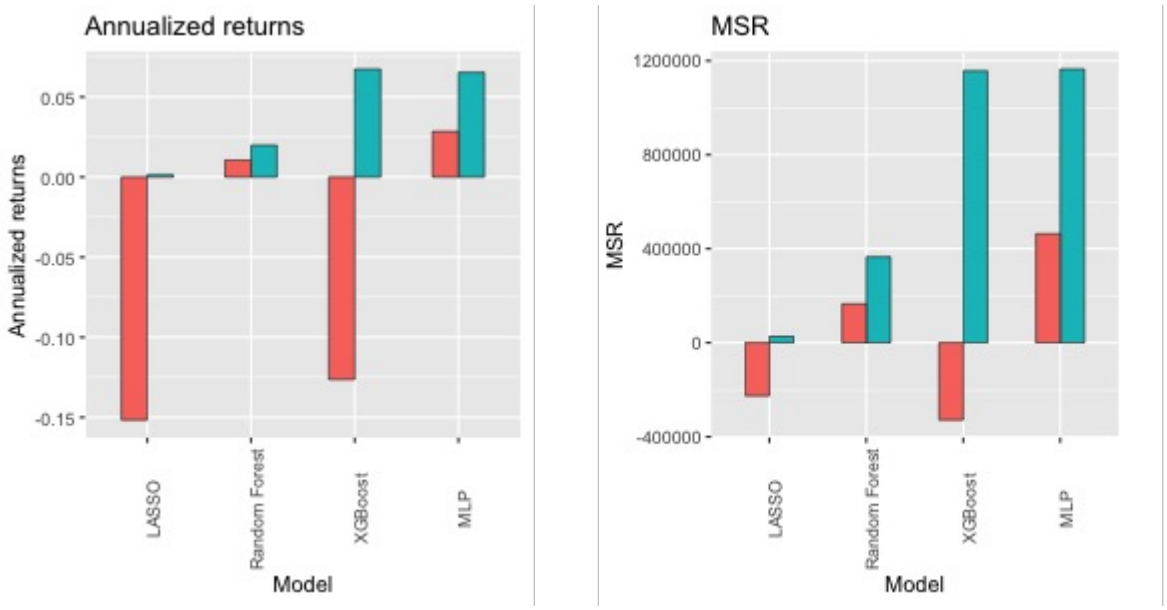

Figure 5.5: Results for $h = 30$

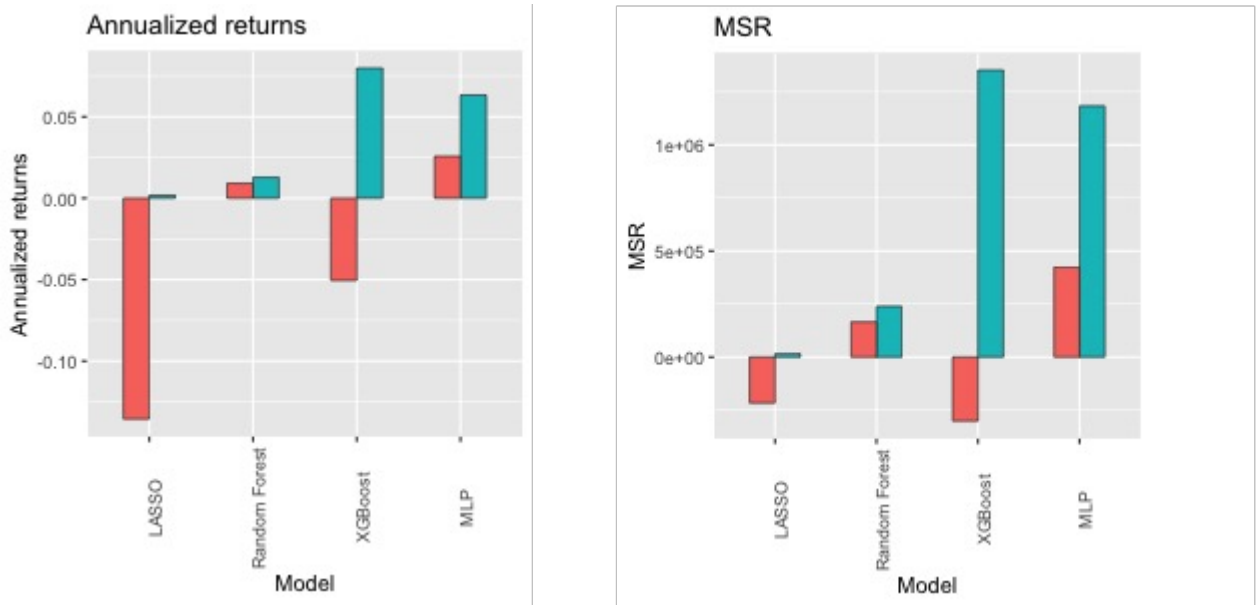

Figure 5.6: Results for $h = 40$

It is very clear that the **Spread Crossing labels are better** as they lead to greater results not only in terms of MSR but also in terms of annualized returns.

Taking this into account, we are going to use the Spread Crossing labels as the target ones with the LSTM network. This makes a lot of sense since, with our trading strategy, when we predict a right spread crossing we are sure that we will make money. On the other hand, we can predict a right mid price movement, buy the asset and still lose money with that purchase.

Finally, we can also conclude from the results that Logistic Regression is not good enough to fit the data, as it performs much worse than the other ones, and some of them even lose money consistently. However, this is no surprise as High Frequency financial data is highly nonlinear, so expecting a linear model to fit it well is unreasonable. Also, Random Forest models significantly underperform XGBoost and MLP models.

For this reason we will discard using LASSO and Random Forest as benchmark, and stop showing its results in the following comparisons.

Now that we have our benchmark models, it is time to build our LSTM model and see if it can outperform them.

Chapter 6

The LSTM model

In this chapter we are going to describe an LSTM cell and the motivations behind it and propose a neural network architecture that we will train to classify the data using the **spread crossing** labels. Finally, we will compare its performance to the benchmarks given in the previous chapter.

6.1 Recurrent neural networks and LSTM

As described in Chapter 5, **recurrent neural networks (RNNs)** are a class of artificial neural networks in which the graph that defines the shape of the network has cycles. Because of this, they process the information sequentially; when producing an output, an RNN takes into account the current and the previous inputs.

This makes RNNs very suitable for dealing with time-series data, especially when we know that the output that we want at a given time step depends partially on the state of the time series in previous time steps.

To simplify the concept of an RNN, we can *unroll* it and think of it as a feed-forward neural network with many layers (at least one layer per time step).

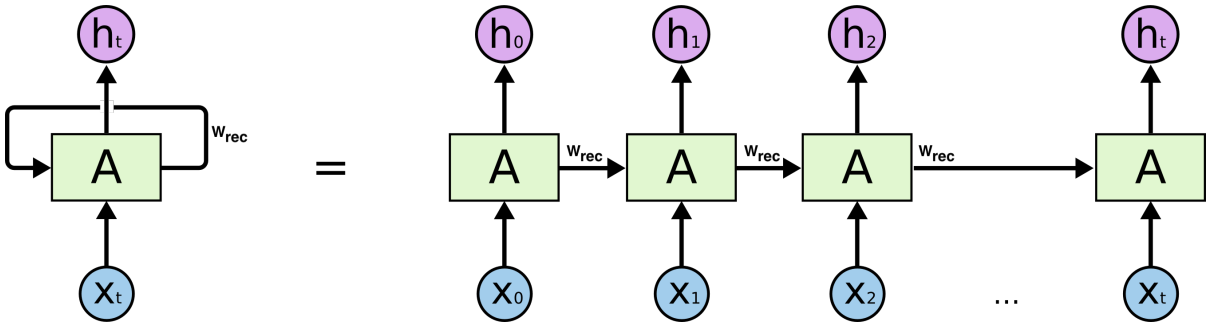

Figure 6.1: Unrolled RNN

Once we realize this, training an RNN can be done in the same way we train a feed-forward network: via gradient descent, using **backpropagation** to compute the gradients. This method is called **backpropagation through time**.

Backpropagation is based on propagating the errors by successively applying the chain rule starting from the output layer and going all the way back to the input one. Now, let's focus on the error term ϵ_t: every neuron that participated in the calculation of the output should have its weight updated in order to minimize that error. In the case of an RNN, it's not just the neurons below this output layer that contributed but all of the neurons far back in time.

The problem with this is that we will successively multiply the gradient by W_{rec} (which is initialized randomly and close to 0) when propagating back in time; this means that the elements of the gradient corresponding to weights that are distant in time will be close to 0, and therefore these weights will not be updated. This is called the **vanishing gradient** problem, and it makes it impossible to the model to learn correlation between temporally distant events [12].

The **Long Short Term Memory (LSTM)** [13] recurrent neural network architecture was designed to overcome this problem. Unlike an standard RNN, in which the repeating module contains one layer, the repeating module of an LSTM contains four interacting layers:

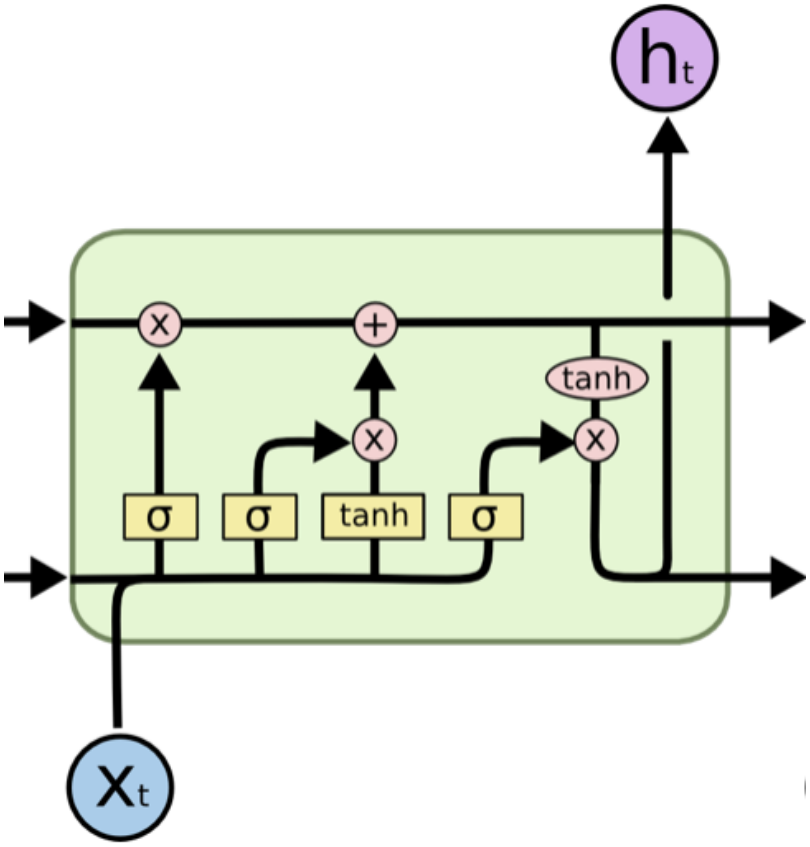

Figure 6.2: Graphical representation of an LSTM cell [14]

The key to LSTMs is the **cell state**, the horizontal line running through the top of the diagram, which is changed by minor linear interactions and makes it very easy for information to flow along time.

Now, let's explain the LSTM functioning step by step. We will refer to the input as x_t, having $x_t \in \mathbb{R}^d$, and to the output as h_t, having $h_t \in \mathbb{R}^h$, with h being the number of hidden units.

1. The first step is to decide what information we are going to throw away from the *cell state*. This decision is made by a sigmoid layer called the *forget gate layer*: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ (σ denotes the sigmoid function).
2. Next we are going to decide what information we are going to store in the *cell state*. This is done in two steps. First, the *input gate layer* decides which values we will update: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$. Next, a tanh layer creates the new

candidates that could be added to the state: $\hat{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$.

3. We now update the old state: $C_t = C_{t-1} \circ f_t + \hat{C}_t \circ i_t$. Here, $\circ$ denotes the Hadamard (element-wise) product.
4. Finally, we use the cell state and the current input to decide what we are going to output; first, we run a sigmoid layer which decides what parts of the cell state we're going to output: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$. Now we apply the tanh function to the cell state to assure that the value is between -1 and 1 and we multiply it by $\varnothing_t$: $h_t = o_t \tanh(C_t)$.

Note that the dimensions of the weight matrices W_f, W_i, W_C, W_o and the weight vectors b_f, b_i, b_C, b_o (which are the parameters that we need to optimize) depend on h, so the more hidden units we have the more complex the model will be.

6.2 Proposed network

As mentioned in the Introduction, our goal is to check if LSTM networks can surpass the need of handcrafting features (as we have done with the benchmark models) and learn directly from the raw financial data. For this reason, we are going to use as input to the model only the raw LOB data:

- **Ask and bid price** at levels 1 to 5.
- **Ask and bid volume** at levels 1 to 5.

In this thesis we propose a network with three LSTM hidden layers. The shape of the network is the following:

- **Input layer**: 20 units (the ask and bid prices and volumes at levels 1 to 5)
- **First LSTM layer**: 16 units
- **Second LSTM layer**: 8 units
- **Third LSTM layer**: 4 units
- **Output layer**: 1 unit (*sigmoid* activation)

As in the benchmark models, the output will be the probability of the inputted sample of belonging to the class *spread cross*.

We will follow the *training framework* described in Chapter 5. As explained in the MLP section, neural networks are trained by minimizing a loss function over the training samples. In our case, we are dealing with binary classification, so we will use the

discrete cross entropy function (also used by XGBoost and MLP models)

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} -(y_i \log(p_i) + (1 - y_i) \log(1 - p_i))$$

where

- N is the number of training samples.
- y_i is the ith sample's class (1 for *spread cross*, 0 for *no crossing*).
- p_i is the model's output.

We will also choose the probability threshold t to optimize the MSR over the validation period.

The network is trained using gradient descent, computing the gradient using the *back-propagation through time* algorithm. The full gradient descent training algorithm can be informally described as following:

Epoch $\leftarrow$ 1
while not converged **do**
  for $i \in [1, ..., \frac{N}{b}]$ **do**
    Gradient = ComputeGradient(training_data$[(i-1) \cdot b + 1, ..., i \cdot b]$)
    Weights $\leftarrow$ Weights $- \lambda \cdot$ Gradient
  end for
  Epoch $\leftarrow$ Epoch + 1
end while

We will use a 1024 *batch size*, which means that at each iteration the gradient is computed using 1024 samples of data, and a learning rate (λ) of 0.1.

An **epoch** finishes once we pass over the whole training data set. Then the next epoch starts, starting from the beginning of the training data. Every time an epoch finishes we will compute the loss over the validation data. This will help us keep up track of how the model is performing: after a certain number of epochs, it is usual for deep learning models to start **overfitting**, which means that it adapts too much to the training data, resulting in a big error when tested with non-training data.

To avoid this, we will consider that the model has converged when the validation loss hasn't improved for 20 straight epochs.

6.3 Results

First we will look at the loss plots for the different time horizons:

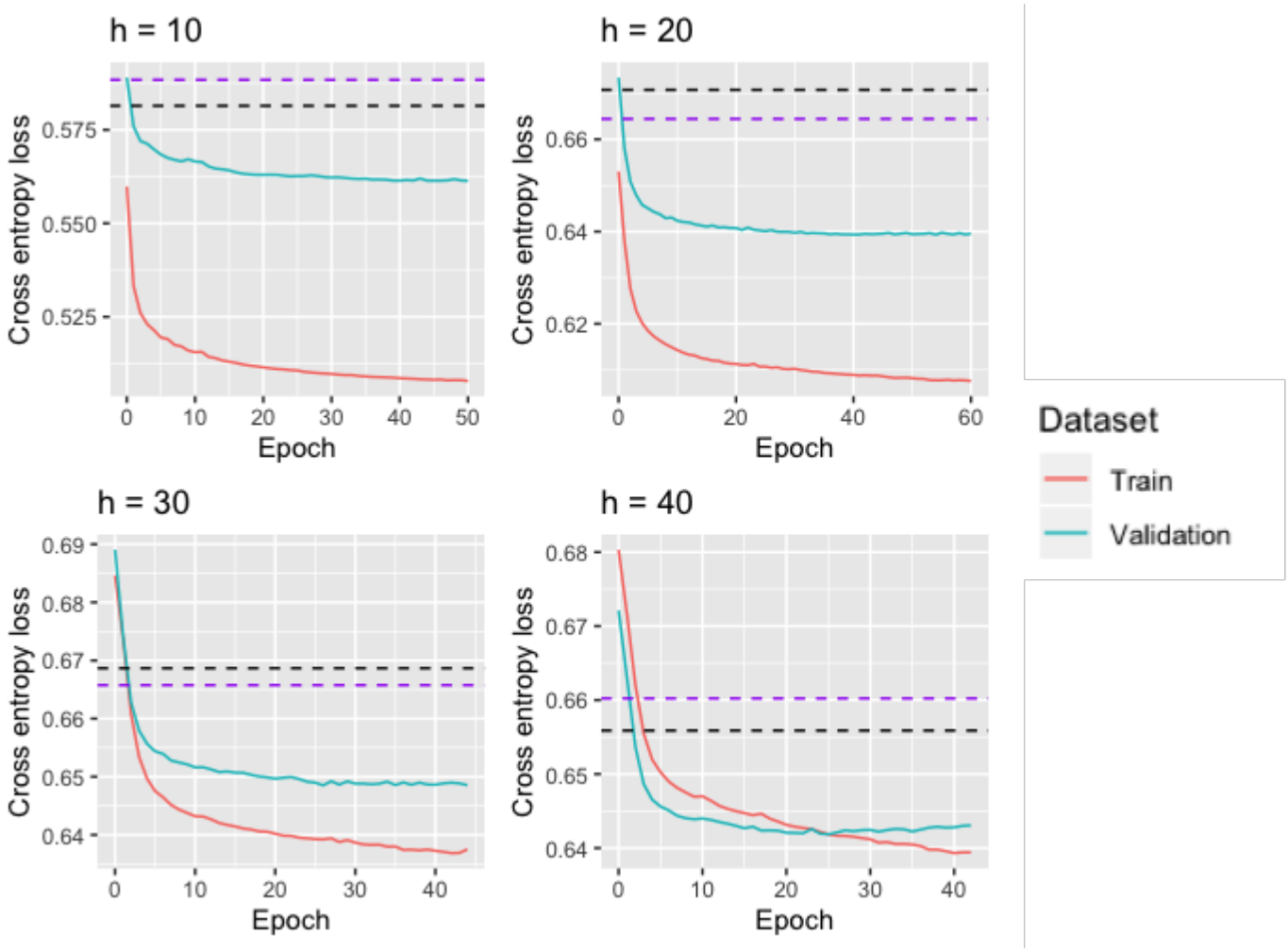

Figure 6.3: Loss plots for the different time horizons. The black and purple dashed lines mark the validation cross entropy for the XGBoost and MLP models respectively

As we can see, with few epochs the LSTM network beats the XGBoost and MLP models, and continues to decrease the loss. This is a really good signal as it means that the LSTM network fits better the data. However, this has to translate into better trading performance.

These are the results after the trading simulation over the test period:

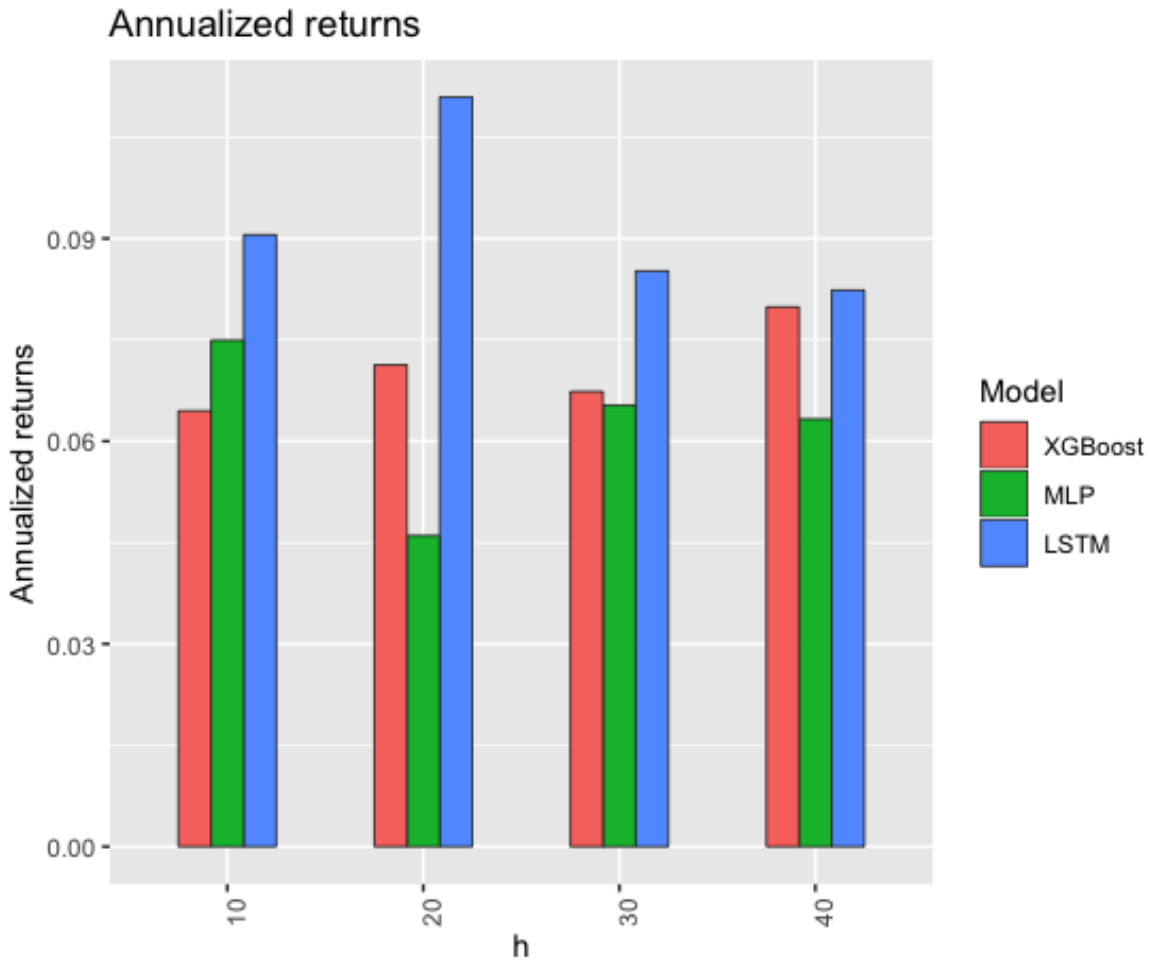

Figure 6.4: Annualized returns

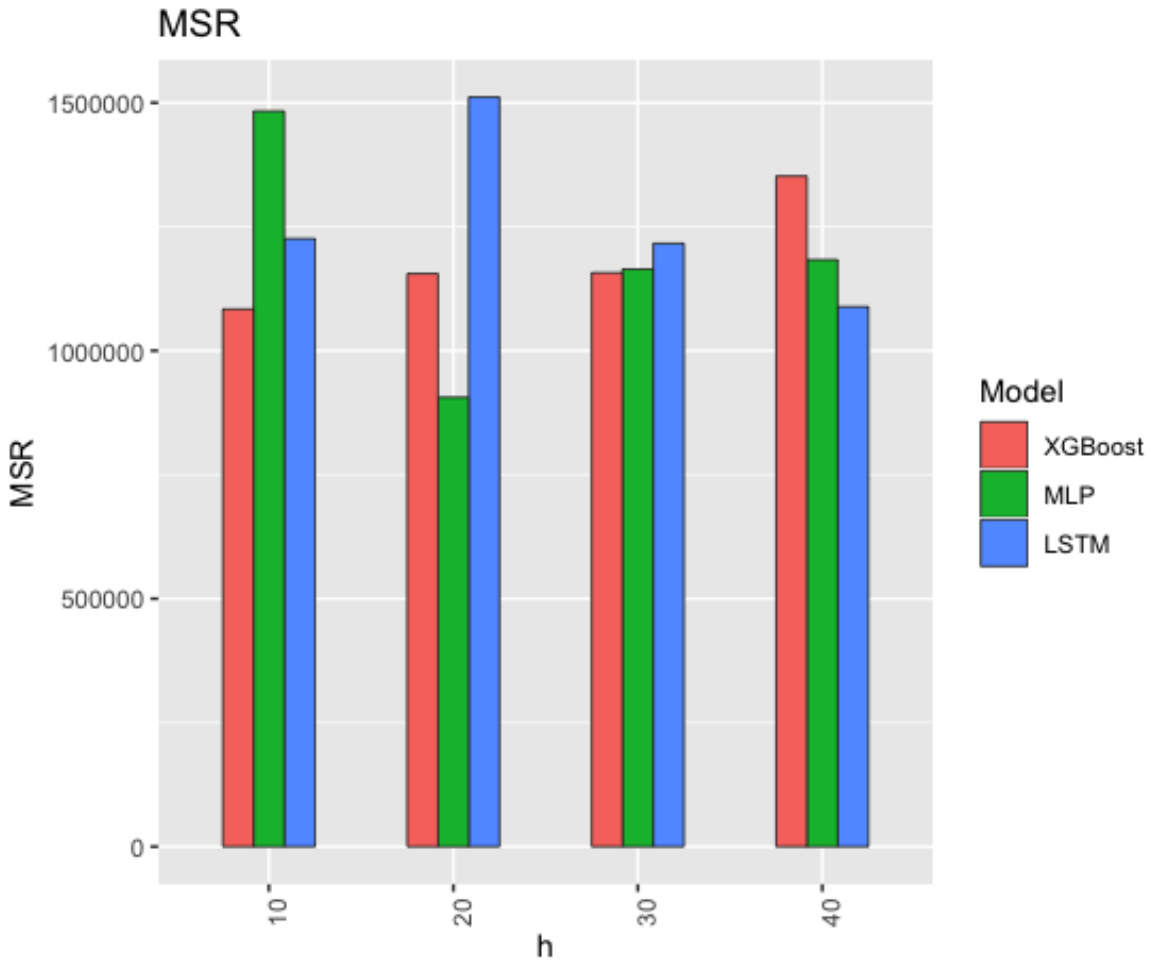

Figure 6.5: Modified Sortino Ratio

In terms of **annualized returns**, the best model is the LSTM one with $h = 20$, followed by the LSTM ones with $h = 30$ and 10.

When looking at the **Modified Sortino Ratio** the best model is also the LSTM one with $h = 20$, but this time followed by the MLP one with $h = 10$ and the XGBoost one with $h = 40$.

As the Modified Sortino Ratio for the MLP model (with $h = 10$) is really close to the one of the LSTM network (with $h = 20$), and the LSTM network beats by far the MLP model when taking annualized returns, this means that the MLP model is more risk-avoidant.

From this results, we can conclude that **the LSTM model with** $h = 20$ **outperforms**

the benchmark ones. However, some risk-averse people would still want to use the MLP one with $h = 10$.

The following graph shows the cumulative returns and drawdown plots for the three best models:

Figure 6.6: Cumulative Return and Drawdown graphs

Looking at this plot we can clearly see how the LSTM model performs better than the other two. We can also see in the drawdown plot how the LSTM model is also more risky as it has greater drawdowns.

Chapter 7

Conclusions

The goal of this thesis was to use a recurrent neural network architecture to deal with raw High Frequency financial data, bypassing the need of handcrafting features from the Limit Order Book data as is done by practitioners nowadays, and making full use of the potential of recurrent neural networks when dealing with time series data. Following this goal, we have presented a recurrent neural network architecture consisting of three LSTM hidden layers and compared it to other state-of-the-art machine learning models that use as input a set of handcrafted features.

Given the results presented in Chapter 6 we can conclude that we have achieved our goal. Firstly, the LSTM models for different horizons are able to get a smaller validation loss than the benchmark models, which means that they are better at fitting the data.

When it comes to trading, it is clear that when dealing with LSTM models the best time horizon is $h = 20$. This model achieves a considerable 11% annualized return, much greater than any of the benchmarks. However, it has a return-to-risk ratio only slightly greater than the MLP benchmark with $h = 10$, which means that some risk-averse people might prefer to use the MLP model.

I also think that it's really important to note how with a really simple trading algorithm and a deep learning model we can achieve an annualized return of 11% with a tiny risk. For reference, if we calculate the traditional Sharpe Ratio for the LSTM model on the testing period we will get an annualized Sharpe Ratio of 33.5! I think that this shows the potential of High Frequency Trading, which is an exciting topic with endless possibilities of improvement and further research.

Finally, I want to thank again Prof. Daniel P. Palomar for giving me this opportunity. I have learnt a lot during these 6 months in Hong Kong, from reading and properly understanding related papers to doing my own research with Prof. Daniel's help any time I needed it. For me, this time has been an introduction to the academic world, and I think that it couldn't have been better.

References

[1] M. D. Gould, M. A. Porter, S. Williams, M. McDonald, D. J. Fenn, and S. D. Howison, "Limit order books," *Quantitative Finance*, Nov. 2013.

[2] N. Hautsch, *Econometrics of financial high-frequency data.* Springer Berlin Heidelberg, 2012.

[3] A. N. Kercheval and Y. Zhang, "Modelling high-frequency limit order book dynamics with support vector machines," *Quantitative Finance*, vol. 15, no. 8, pp. 1315–1329, 2015, doi: 10.1080/14697688.2015.1032546.

[4] A. Ntakaris, G. Mirone, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Feature engineering for mid-price prediction with deep learning," *IEEE Access*, vol. 7, pp. 82390–82412, 2019, doi: 10.1109/access.2019.2924353.

[5] P. Nousi *et al.*, "Machine learning for forecasting mid price movement using limit order book data." 2018.

[6] A. Ntakaris, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods," 2017, doi: 10.1002/for.2543.

[7] Z. Zhang, S. Zohren, and S. Roberts, "DeepLOB: Deep convolutional neural networks for limit order books," 2018, doi: 10.1109/TSP.2019.2907260.

[8] D. P. Palomar, "Portfolio optimization with r." MSc in Financial Mathematics, HKUST, [Online]. Available: https://www.danielppalomar.com/mafs6010r---portfolio-optimization-with-r.html.

[9] H. Markowitz, "PORTFOLIO SELECTION," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, Mar. 1952, doi: 10.1111/j.1540-6261.1952.tb01525.x.

[10] W. F. Sharpe, "Mutual fund performance," *The Journal of Business*, vol. 39, no. S1, p. 119, Jan. 1966, doi: 10.1086/294846.

[11] F. Sortino and L. Price, "Performance measurement in a downside risk framework," *Journal of Investing.*.

[12] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks." 2012.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997, doi: 10.1162/neco.1997.9.8.1735.

[14] C. Olah, "Understanding lstm networks." [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.